Community Embeddings for Friend Suggestions

Anton I. N. Begehr
Graduate School of Business
National Research University Higher School of Economics
Moscow, Russia
a.begehr@fu-berlin.de

Prof. Dr. Petr Panfilov
Graduate School of Business
National Research University Higher School of Economics
Moscow, Russia
ppanfilov@hse.ru

***Abstract*—Graphs, such as social networks, emerge naturally from various real-world situations. Recently, graph embedding methods have gained traction in data science research. Recommender systems are used in a wide range of business applications and are essential for online, e-business models to survive and strive in the contemporary market. Using graph embeddings for recommendation tasks, have the possibility of improving upon recommender systems, because of data compression, their feature vector format, and sub-quadratic time complexity. Graph and community embeddings generated with ComE BGMM+VI are used to build a recommender system for friend suggestions. ComE BGMM+VI is an alteration of the community embeddings algorithm ComE. ComE BGMM+VI applies a Bayesian Gaussian mixture model and variational inference for community embedding and detection. Recommendations are evaluated by the top-N hit-rate over users with at least 50 friends. A friend suggestions recommender system with a top-10 leave-one-out hit-rate of 43.6% and run-time optimized 32.9% is presented.**

***Index Terms*—graph, embedding, community embedding, ComE, recommendations, friend suggestions**

I. Introduction

Graphs, such as social networks, knowledge graphs, content-rating graphs, and communication networks, emerge naturally from various real-world situations. Analyzing these graphs leads to findings and understanding of the underlying structures, coherences, and dependencies. Recently, methods for embedding graph's nodes into lower-dimensional Euclidean spaces, called graph embeddings, have gained traction in multiple areas of data science research [**Goyal˙2018**].

Due to the rapid growth of the internet and data accumulation, recommender systems are essential for e-business and online business models to survive and strive in the contemporary market [**Polatidis2013RecommenderST**]. Modern recommender systems need to take into account the huge amounts of user data generated at all times in big data systems around the world and improve recommendations instead of failing under the thrust of big data overload.

Utilizing graph embeddings for recommendation tasks, has recently gained research traction [**Palumbo2018AnEC**, **Palumbo2018KnowledgeGE**, **GradGyenge2017GraphEB**, **Sathish2019GraphEB**]. The advantages of graph embeddings include data compression and the Euclidean feature vector format [**Godec2018**]. Given these advantages and provided competitive results, graph embeddings have the possibility of greatly improving upon graph-based use-cases like recommender systems.

Community Embeddings, in addition to embedding a graph's nodes through first- and second-order proximity, also preserve higher-order proximity by embedding clusters present in the graph data. The graph and community embedding algorithm ComE aims to preserve first-, second- and higher-order proximity by embedding a graph's nodes and communities [**ComE**].

This work specifically examines community embeddings for friend suggestion recommender systems and evaluates recommendations on social network graph data for the use-case of friend suggestions. Graph and community embeddings generated with ComE BGMM+VI are used to develop a friend suggestions recommender system based on the shortest distances between nodes in the embedding. Recommendations are evaluated by the top-N recommendations hit-rate of test edges. A friend suggestions recommender system with a top-10 leave-one-out hit-rate of 43.6% and run-time optimized 32.9% is presented.

II. Friend Suggestions

Recommender Systems are eagerly researched in academia and widely deployed in real-world business applications. Most contemporary technology companies heavily rely on recommender systems to drive usage of their services and consumption of their content. Users, in turn, rely on recommender systems to find what they want and need and save searching time. State-of-the-art recommender systems provide a competitive advantage desperately needed by online services. Companies heavily relying on recommender systems include YouTube, Amazon, Netflix, and many more [**Rocca19**].

Recommender systems are built on top of user-item interaction. Friend suggestions are a simpler type of recommender system. For friend suggestions, no distinction is made between users and items. The entity user is both the subject and object of recommendation. This results in a simple data model, which can be used for structural friend suggestions, as shown in Fig. **??**:

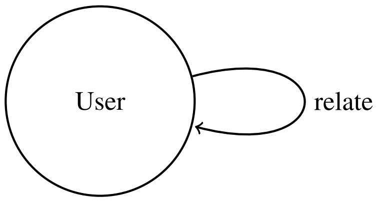

Fig. 1. Minimal data model supporting friend suggestions.

The two main methods used for recommender systems are collaborative filtering and content-based filtering. Collaborative filtering is based on the assumption that users like items similar to other items they like and items that are liked by other users with similar tastes [**Grover17**, **Su2009ASO**]. Content-based filtering considers user and item attributes, instead of solely interactions [**Rocca19**, **Lops2011SurveyCB**].

In this paper, a method of generating friend suggestions based solely on graph data structure, and not on the node or edge attributes, is presented. The graph of users and friendships is embedded to a lower-dimensional space with the community embedding algorithm ComE BGMM+VI [**ComE**, **ComE+**, **ComE˙BGMM˙GH**]. The approach can be considered a variant of collaborative filtering with the euclidean distance of user embeddings and user community membership as measures for user similarity. The proposed advantage of using such an approach is that community embeddings optimize first-, second-, and higher-order proximity between users in the node embedding.

A. Evaluation

To determine the effectivity of friend suggestions, the generated friend suggestions must be properly validated. In this paper, the top-N approach to evaluating recommender systems with hit-rate as the evaluation metric is chosen.

A user's top-N recommendations is a list of N items to be recommended to a specific user. To evaluate recommender systems with the top-N and hit-rate approach, initially, the dataset is split into train and test data. For each testing user, one relation is left out, according to the leave-one-out method and the model is trained on the remaining training dataset. Once the model is trained, a list of top-N recommendations is generated for each testing user. If the item corresponding to the user is in the user's top-N list, a hit is counted, otherwise, a miss is counted. The hit-rate is defined as the total number of hits divided by the number of testing users.

Utilizing the top-N approach with hit-rate to evaluate recommender systems is advantageous to evaluating recommender systems by a link prediction approach since the top-N approach is more realistic in comparison to actual recommender system use-cases. When you open Netflix, Amazon, YouTube, or Facebook friend suggestions, one or multiple top-N recommendation lists are generated and displayed. If you click on an item and buy, watch, or befriend, that is considered a hit, otherwise, a miss.

The hit-rate metric on top-N recommendations provides a realistic option of evaluating recommender systems [**Cremonesi2010PerformanceOR**, **Palumbo2017entity2rec**, **Zhao2019ATS**].

III. Algorithm

The proposed algorithm for generating friend suggestions using community embeddings is detailed and evaluated on time complexity. The initially quadratic runtime of generating friend suggestions for all users is then reduced by a factor of K by utilizing a node's community membership. The two resulting algorithms are presented.

Algorithm 1 Top-N Social Recommendations based on Node Embeddings

```
Require: graph $G = (V, E)$, maximum number of communities K, number of walks $\gamma$, walk length $\ell$, window size $\zeta$, representation size $D$, negative context size $m$, parameters ($\alpha$, $\beta$), number of recommendations $N$.
Ensure: Top-$N$ recommendations for all nodes $R$.
 1: $\Phi, \Phi', \Pi, (\Psi, \Sigma) \leftarrow ComE_{BGMM+VI}(G, K, \gamma, \ell, \zeta, D, m, \alpha, \beta)$ ▷ Run ComE BGMM+VI.
 2: for $(v, v') \in E$ do ▷ Determine sets of friends for each user.
 3:     $F_v \leftarrow F_v \cup \{v'\}$
 4:     $F_{v'} \leftarrow F_{v'} \cup \{v\}$
 5: end for
 6: for $v \in V$ do
 7:     $R_v \leftarrow SortedDict(size = N)$ ▷ Init recommendations for node $v$.
 8:     for $v' \in V \wedge v' \notin F_v \wedge v' \neq v$ do ▷ For all non-friends of user $v$.
 9:         $d \leftarrow \|\phi_v - \phi_{v'}\|$ ▷ Distance of $v$ and $v'$ embeddings.
10:         $R_v[d] \leftarrow v'$ ▷ Add $v'$ to $v$'s recommendations at distance $d$.
11:     end for
12:     $R_v \leftarrow R_v.values()$ ▷ Get top-$N$ recommendations for each node.
13: end for
```

Algorithm **??** describes in pseudocode how recommendations are computed from node embeddings generated by ComE BGMM+VI. The terms friend suggestions and social recommendations are used interchangeably.

The referenced function in line 1 $ComE_{BGMM+VI}(G, \gamma, \ell, \zeta, D, m, \alpha, \beta)$ is the ComE BGMM+VI algorithm as described in Section **??**. ComE BGMM+VI is based on the ComE algorithm presented by **ComE** in their paper **ComE** as algorithm 1 on page 381 of the publication's in-proceeding. The runtime of ComE BGMM+VI is equivalent to the runtime of ComE in it's big-O notation. ComE's time complexity is $O(V\gamma\ell + V + T_1(T_2VK + K + E + V\gamma\ell + VK))$, which is linear in time complexity to the graph size: $O(V + E)$.[**ComE**] T_1 is the number of iterations of ComE's closed loop consisting of node embedding, community detection, and community embedding to run. T_2 is the number of iterations to run expectation-maximization or variational inference for community detection and embedding.

ComE uses a Gaussian mixture model and expectation-maximization for community embedding, therefore heavily relying on the number of communities K hyperparameter for embedding quality. ComE BGMM+VI instead takes a Bayesian approach by utilizing a Bayesian Gaussian Mixture with variational inference for community embeddings. variational inference allows dropping unused communities and therefore is less dependent on the number of communities K hyperparameter. In their paper **ComE+**, **ComE+** present a Bayesian approach to ComE as algorithm 45 of their publication's magazine. ComE+'s time complexity is $O(V\gamma\ell + V + T_1(T_2VK + K + E + V\gamma\ell + VK))$ as well, which is linear in time complexity to the graph size: $O(V + E)$.[**ComE+**]

The function $ComE_{BGMM+VI}(G, K, \gamma, \ell, \zeta, D, m, \alpha, \beta)$ returns the node embedding Φ, context embedding Φ', community assignment Π, and community embedding (Ψ, Σ).

Let us consider the full algorithm's time complexity now:

1) Line 1: The node and community embeddings are generated with ComE BGMM+VI in linear time to the graph size: $O(V\gamma\ell+V+T_1(T_2VK+K+E+V\gamma\ell+VK)) = O(VE)$.
2) Lines 2-5: All friends of each user are determined in $O(E)$.
3) Lines 6 through 13 compute a dictionary of the Top-N friend suggestions for each user sorted ascending by distances between the node embeddings. Each node is considered: $O(V)$.
    a) Line 7: A SortedDict is initialized in $O(1)$.
    b) Lines 8-11: Iterating over all non-friends v' of user v is achieved in $O(V)$. The distance in embedding between user v and v' are calculated in line 9 in $O(1)$. In line 10, friend suggestions are added to user v's sorted friends suggestions in $O(N)$. The number of friends per user $\overline{F_v}$ is not considered, because $\overline{F_v} \ll V$. Lines 8-11 are therefore achievable in $O(VN)$.
    c) Line 12: Friend suggestions are returned as lists of top-N users in $O(1)$.

    Lines 6 through 13 are achieved in $O(V^2N))$

According to the analysis in the list above, the total time complexity of Algorithm **??** is $O(V\gamma\ell+V+T_1(T_2VK+K+E+V\gamma\ell+VK)+E+V^2N)$. We consider γ, ℓ, T_1, T_2, and N as constant, therefore the resulting time complexity in big-O notation depends on the graph's size (V, E) and the number of communities K: $O(K+VK+E+V^2N)$. The number of communities K can also be considered as constant, in which case the time complexity is $O(E+V^2N)$.

Algorithm 2 Top-N Social Recommendations based on Node and Community Embeddings

```
Require: graph $G = (V, E)$, maximum number of communities K, number of walks $\gamma$, walk length $\ell$, window size $\zeta$, representation size $D$, negative context size $m$, parameters ($\alpha$, $\beta$), number of recommendations $N$.
Ensure: Top-$N$ recommendations for all nodes $R$.
 1: $\Phi, \Phi', \Pi, (\Psi, \Sigma) \leftarrow ComE_{BGMM+VI}(G, K, \gamma, \ell, \zeta, D, m, \alpha, \beta)$ ▷ Run ComE BGMM+VI.
 2: for $v \in V$ do   ▷ Generate nodes sets $C_k$ per community $k$.
 3:     $k \leftarrow \pi_v$
 4:     $C_k \leftarrow C_k \cup \{v\}$
 5: end for
 6: for $(v, v') \in E$ do   ▷ Determine sets of friends for each user.
 7:     $F_v \leftarrow F_v \cup \{v'\}$
 8:     $F_{v'} \leftarrow F_{v'} \cup \{v\}$
 9: end for
10: for $v \in V$ do
11:     $R_v \leftarrow SortedDict(size = N)$   ▷ Init recommendations for node $v$.
12:     for $v' \in C_k \wedge v' \notin F_v \wedge v' \neq v$ do   ▷ For all non-friends of user $v$.
13:         $d \leftarrow \|\phi_v - \phi_{v'}\|$   ▷ Distance of $v$ and $v'$ embeddings.
14:         $R_v[d] \leftarrow v'$   ▷ Add $v'$ to $v$'s recommendations at distance $d$.
15:     end for
16:     $R_v \leftarrow R_v.values()$   ▷ Get top-$N$ recommendations for each node.
17: end for
```

Algorithm **??** describes in pseudocode how recommendations are computed from ComE BGMM+VI node and community embeddings and community assignments. The advantage of the second variant of the algorithm, which also considers communities, is that an improvement in time complexity can be obtained, by considering only users in the same community for friend suggestions.

Line 1, lines 6-9, and lines 10-17, except for a small edit in line 12, are the same as in Algorithm **??**. Differences to Algorithm **??**:

1) Lines 2-5: All users are filtered into sets C_k, one for each community k in $O(V)$.
2) Line 12: Instead of considering the set of all non-friends users of size V, only users in the same community C_k of size $\frac{V}{K}$ are considered. Lines 10 through 17 are achievable in $O(\frac{V^2N}{K})$.

According to the analysis from Algorithm **??** and the differences to Algorithm **??** above, the total time complexity of Algorithm **??** is updated to $O(V\gamma\ell+V+T_1(T_2VK+K+E+V\gamma\ell+VK)+V+E+\frac{V^2N}{K})$. Again, we consider γ, ℓ, T_1, T_2, and N as constant, therefore the resulting time complexity in big-O notation depends on the graph's size (V, E) and the number of communities K: $O(K + VK + E + \frac{V^2N}{K})$. The number of communities K can also be considered as constant, in which case the time complexity is $O(E + V^2N)$.

The time complexity of Algorithm **??**, when not considering K as a substantial factor in time complexity is equivalent to the time complexity of Algorithm **??**. Nevertheless, it is clear, that K can substantially reduce the quadratic summand V^2N through division by only considering users of the same community for friend suggestions.